

OCR Advanced GCE in Computer Science

H446-03/04

Unit 3

Project Advice

(For computer games only)

H446-03/04 – PROJECT ADVICE

CONTENTS

Forward by craig'n'dave.....	5
General Comments.....	6
Overview.....	6
Adopting the agile approach to software development.....	6
Choice of programming language.....	7
project ideas.....	7
The user.....	8
About this document.....	8
Analysis (Max 10 marks).....	9
Description of the problem.....	10
An outline of the project.....	10
Stakeholders.....	11
Identification of all the stakeholders.....	11
Identification of the target platform.....	11
Explanation of the user needs.....	12
Research.....	12
Detailed research into existing similar products.....	12
Features of the proposed solution.....	12
Explanation and justification of features of your product.....	12
Identification of limitations and scope.....	13
Computational methods.....	13

Abstraction and visualisation.....	13
Thinking ahead.....	13
Thinking procedurally.....	13
Thinking logically.....	14
Thinking concurrently.....	14
Hardware and software requirements.....	14
Identification of the hardware requirements.....	14
Identification of the software requirements.....	15
Identification of utilities/libraries.....	16
Success criteria.....	16
Identification of the success criteria (requirements specification).....	16
Evidence Do's and don'ts checklist.....	17
Design (Max 15 marks).....	18
Structure of the solution.....	19
An overview of the structure of the solution (a systems diagram).....	19
Decomposition of the problem.....	20
Explanation of each of the modules, methods, procedures and functions required.....	20
A justification of the approach taken.....	21
Key variables and structures.....	21
Identification of the key classes and attributes if object oriented methods are used.....	21
Explanation and justification of the data structures: arrays/lists and files.....	23
Algorithms.....	24
A set of algorithms in pseudocode to describe each of the methods/functions.....	24
Usability features.....	26
Explanation and justification of the design of the user interface.....	26
Description and justification of the usability features.....	27

Description and justification of the validation required.....	27
Test data for development.....	29
Identification and justification of test data to be used during development.....	29
Test data for alpha testing.....	30
Identification and justification of the test data to be used post-development to ensure the system meets the success criteria.....	30
Identification of data that is designed to test the robustness of the solution.....	30
Evidence Do's and don'ts checklist.....	31
Developing a coded solution (Max 25 marks).....	32
Iterative development of a coded solution (Max 15 marks).....	32
Testing to inform development (Max 10 marks).....	33
Iterative development.....	34
Evidence Do's and don'ts checklist.....	38
Evaluation (Max 20 marks).....	39
Testing to inform evaluation (Max 5 marks).....	39
Testing.....	39
Test table.....	39
Video evidence.....	40
Usability features.....	40
Illustration of how the usability features have been tested to make sure they meet the user needs.....	40
Evaluation of solution (Max 15 marks).....	41
Evaluation.....	42
Commentary on how well the solution matches the requirements.....	42
Evidence Do's and don'ts checklist.....	43
project hand in checklist.....	44
INITIAL PROJECT IDEAS.....	45

FORWARD BY CRAIG'N'DAVE

This project advice is designed to be a one-stop-shop for all the guidance you will need to complete your unit 3 programming project.

We have done the hard work, we have been through all the advice, tips and guidance documents from OCR, read in detail past examiner reports, looked through numerous text books, read up on clarification documents and distilled all this information into one handy place: THIS DOCUMENT!

We also bring our many years of personal experience from our own classrooms to this document, we know what makes a successful project, and we know what advice to give our own students to make sure they are best equipped to get the top marks.

In the first time through on this new qualification both myself and Dave received very positive praise on our marking of the projects and the marks we submitted were unaltered:

"The marks were considered to be in line with the national standard. Full credit to the centre for a professional performance with the first attempt at a new specification."

(Craig's moderator feedback)

"The centre have appreciated the requirements and are able to apply them realistically."

(Dave's moderator feedback)

We hope these comments gives you confidence in this document, read it all, follow its advice carefully and you will find yourself fully supported during your project.

In addition, make sure to check out our play list titled "A level: OCR Unit 3 Project Advice" on our CraignDave YouTube channel which contains a series of videos providing additional advice and guidance.

Note: This document has been written specifically with the aim of helping students who are producing computer game programming projects. If you are producing any other type of project use our other version of the project advice.

Best of luck!

Craig & Dave

GENERAL COMMENTS

OVERVIEW

The programming project is a major part of the A'Level course. It is worth 20% of your final grade.

You are required to demonstrate your ability to analyse, design, develop, test, evaluate and document a complete program written in a suitable programming language (see below for a list of languages you are allowed to choose from).

It is important that the nature of the problem you choose to solve allows you to demonstrate the full range of skills and techniques required in the mark scheme. Trivial problems, regardless of how well you solve them and write them up will not be able to provide you the right evidence (there is guidance below on the appropriate project ideas). Computer games are ideal because they have plenty of opportunities for a full range of computational thinking and methods as well as being fun to make!

ADOPTING THE AGILE APPROACH TO SOFTWARE DEVELOPMENT

It is stated by the exam board that you are expected to “Apply the computational approaches identified in Unit 2 to a practical coding problem and apply the principles of an agile development approach to the project development.”

This means that development of a solution is an iterative process. You will tackle each part of your problem in turn, coding a procedure, module or function, test it, modify it then moving onto the next part. During the process of development you will regularly get feedback from your target market. They will provide comments on how your solution is developing. It is very important to capture the outcome of those discussions in your report and show how your ideas are developing as a result over time.

During development, due to problems, issues highlighted from ongoing testing or simply due to feedback from your user you might discover omissions or problems with your original requirements or design work. **DO NOT** go back and alter your requirements or designs to match, this is perfectly natural during development and the examiner expects to see this. Simply record any changes, new requirements, new algorithms, new tests in your development section as they appear.

This extended development section thus becomes a narrative on the process of producing your solution. This is known as “telling the development story”, it is the most natural way to capture evidence and is exactly what the examiners are expecting to see.

This will mean that evidence to support assessing your project might be found throughout your project report.

CHOICE OF PROGRAMMING LANGUAGE

The choice of which programming language to choose for your project is not a simple one. It will make sense to choose a language you are comfortable with, so we would suggest using the one you have been predominantly learning to program in, or one similar to it. Your teacher / tutor will also be able to guide you in a choice of language for your project.

Whatever language you choose the exam board state that ***“All tasks completed in all languages need to have a suitable graphical interface”***.

They go on to specify the programming languages OCR will accept. These are:

- Python
- C family of languages (C# C+ etc.)
- Java
- Visual Basic
- PHP
- Delphi
- Monkey-X (Also now approved by OCR and excellent for games development)

This list should allow you to develop most programs you have in mind. For example, if you wanted to create a mobile phone app for an Android phone this could be done in Java. If you wanted to create it for an iPhone this could be done in Object C.

If you would like to use a programming language not on the list above, it is essential that your teacher / tutor contacts OCR and uses the consultancy service to get approval.

Note: Simple programming environments often used to introduce programming at KS3 such as Kodu, Scratch, Gamemaker are not appropriate for A Level.

PROJECT IDEAS

Your choice of program for your project is one of the most important choices you will have to make. Essentially you can solve any problem you wish, however it is vitally important that you get the scope and level of complexity right.

When deciding on your choice of game, think carefully about the complexity involved. Board games that have fewer moving objects are often easier than those that contain sprites. Games where all the action is limited to one screen are easier than side scrollers. Shoot 'em ups are generally easier than platform games. You should definitely attempt a 2D retro game such as space invaders, Pacman, breakout, Tetris, Asteroids or board games such as Monopoly rather than 3D games which are significantly harder.

The problem you choose to solve must be sufficiently complex that you will be able to meet all the requirements of the mark scheme, but not overly ambitious so as to become unachievable. Remember

this is only A'level. You will not be writing Call of Duty! It is always better to have a small program well done and well written up than a huge problem you can't solve.

Ultimately your teacher / tutor will have to approve your choice of problem to solve.

THE USER

Computer games are written for a specific target audience. Developers will review and evaluate previous games in the franchise or genre before developing a new game. They will seek feedback from the community before, during and after the development process. This often involves having a small number of consultants from the community and a larger body of alpha testers. With smaller studios, there may be a pre-alpha release in which open dialogue from the community is encouraged, and the game is developed based on feedback from the community. With AAA titles the details of the game are often a closely guarded secret, but there will still be a group of play-testers who will sign non-disclosure agreements and there may be a beta-testing phase before a final release.

You will need to take this approach with your project by identifying a broad target market, and then an individual, or small number of individuals who will play an active part throughout the development process, shaping the final product. **To gain high marks it is essential that you fully document the involvement of the user throughout your project.**

ABOUT THIS DOCUMENT

What follows is the mark scheme for each section of the report write-up taken word for word from the specification (in red text). Following this is additional advice and guidance from us (in black text).

Small samples of work for some sections are presented in green text.



Also keep a close eye out for this icon in the margin and make sure to read all the advice next to it carefully (in blue text). Without carrying out all of these actions you will be unable to access to highest marks.

Not all projects will necessarily easily fit the mark scheme so these comments inform you how to tackle each section.

Each section also comes with an evidence Do's & Don'ts checklist.

ANALYSIS (MAX 10 MARKS)

1-2 marks

- Identified some features that make the problem solvable by computational methods.
- Identified suitable stakeholders for the project and described them and some of their requirements.
- Identified some appropriate features to incorporate into their solution.
- Identified some features of the proposed computational solution.
- Identified some limitations of the proposed solution.
- Identified some requirements for the solution.
- Identified some success criteria for the proposed solution.

3-5 marks

- Described the features that make the problem solvable by computational methods.
- Identified suitable stakeholders for the project and described how they will make use of the proposed solution.
- Researched the problem looking at existing solutions to similar problems identifying some appropriate features to incorporate into their solution.
- Identified the essential features of the proposed computational solution.
- Identified and described some limitations of the proposed solution.
- Identified most requirements for the solution.
- Identified some measurable success criteria for the proposed solution.

6-8 marks

- Described the features that make the problem solvable by computational methods and why it is amenable to a computational approach.
- Identified suitable stakeholders for the project and described them and how they will make use of the proposed solution and why it is appropriate to their needs.
- Researched the problem in depth looking at existing solutions to similar problems identifying and describing suitable approaches based on this research.
- Identified and described the essential features of the proposed computational solution.
- Identified and explained any limitations of the proposed solution.
- Specified the requirements for the solution including (as appropriate) any hardware and software requirements.
- Identified measurable success criteria for the proposed solution.

9-10 marks

- Described and justified the features that make the problem solvable by computational methods, explaining why it is amenable to a computational approach.
- Identified suitable stakeholders for the project and described them explaining how they will make use of the proposed solution and why it is appropriate to their needs.
- Researched the problem in depth looking at existing solutions to similar problems, identifying and justifying suitable approaches based on this research.
- Identified the essential features of the proposed computational solution explaining these choices.
- Identified and explained with justification any limitations of the proposed solution.
- Specified and justified the requirements for the solution including (as appropriate) any hardware and software requirements.
- Identified and justified measurable success criteria for the proposed solution.

In this section you are discussing **what** game you are making.

It should set the scope of your project. What it is going to be, who the users are, and detailed research into similar games. How that influences decisions to be made. It is an in-depth look at what will be required and the reasons why.

The analysis is one of the most important parts of your project, but it can also be one of the most difficult to get going with. Read the general advice below and then what you should write about in each section before you start.

Analysis overview:

1. Identify the game you are making: the background to the origins of the game.
2. Identify the player base (stakeholders) and expectations of the genre. Think about when and how they will play the game.
3. Research every screen/object/detail of the original game.
4. Research other adaptations of the game, or similar games in the genre to see how the game has been adapted in different implementations.
5. From this synthesise the research to decide the features you will include or not include. Make decisions on the core game mechanics.
6. From this identify the computational methods: outline abstraction. Thinking procedurally, logically and ahead.
7. Having done all this, you will now know the hardware and software that will be required to both create and play the game.
8. State the success criteria.

Strongest projects would include:

DESCRIPTION OF THE PROBLEM

AN OUTLINE OF THE PROJECT

For games based projects you will want to set the scene so the examiner is familiar with the genre of your game with examples of similar games in the genre.

A statement naming the project, and what type of project it is would be a good idea. You can then explain in more detail what the expectations of the genre are.

A list of computer game genres can be found at:

https://en.wikipedia.org/wiki/List_of_video_game_genres

IDENTIFICATION OF ALL THE STAKEHOLDERS

Unlike other projects, with computer games there isn't one single user, but a target market. Identifying a specific person that can represent this target market to give you feedback before, during and after the development process is a good idea.

The target market will influence your design. For example, if you were developing a game for very young children, you may use bright primary colours, large sprites etc. There may be certain conventions in your genre to consider, and expectations from the target market. For example, gamers today might expect a long campaign story mode, a large range of different weapons etc.

Consider the PEGI rating system: <http://www.pegi.info/en/index/id/33/>

For example, in a game rated "3" the child should not be able to associate the character on the screen with real life characters, they should be totally fantasy. The game should not contain any sounds or pictures that are likely to scare or frighten young children. No bad language should be heard.

You should have a general audience, and an identified person representing that audience who will give you feedback.

It is important to explain how they will use the game. Have you seen the promotional video for the Nintendo Switch? This will give you ideas about what this means.



For the highest marks in this section make sure not just simply list our users / stakeholders. Make sure to explain how they will make use of your proposed game and explain why it is suitable for them.

IDENTIFICATION OF THE TARGET PLATFORM

Games are either exclusive to, or available on a range of platforms. A platform might be the PC market, the Xbox or PlayStation. Each platform has different hardware and capabilities. When games are developed they are targeted to these platforms, but this can impact on the design decisions.

For example, the use of files is very different on web-based games compared to desktop games. Console games do not make use of a keyboard and mouse. On mobile platforms you may be able to use motion sensing.

Prototypes will often use alternative control systems that can easily be changed before building the final game code. An example might be detecting a mouse click to simulate a finger tap on a mobile phone.

EXPLANATION OF THE USER NEEDS

Different types of games have very different play styles. For example, games on a smart phone are usually short: pick them up, play, put them down. Many of the most popular games such as Candy Crush are simple puzzles that are easy to learn, quick to play, and get progressively more difficult. Games on a PC can often be significantly more in depth requiring many hours of play.

In some games the player "levels up", increasing the capability of their character. They may also collect trophies or similar rewards for completing aspects of the game.

RESEARCH

DETAILED RESEARCH INTO EXISTING SIMILAR PRODUCTS

Examining other similar games in the genre provides inspiration and ideas, and are a talking point for a discussion with the target user later. It is common in the games development industry for developers to brainstorm what they like and dislike about similar games and previous games in a series. They listen to community feedback, and think about where the industry is heading.

A good approach to analysing existing games is to think about every object in the game and ask yourself: what does it look like? How does it move? How does it behave?

Write this up for the original game, or the closest version of the game you are making in detail.

Now explore other similar games. How have they adapted the original concept? You need a collection of different ideas on the same game to inform the next section of the write-up where you will make decisions on what to include, and what to not include.



For the highest marks make sure that at the end of your research you identify **AND** justify which approach you are going to take towards your project. The approach you take should be related back to the research you have just carried out.

FEATURES OF THE PROPOSED SOLUTION

EXPLANATION AND JUSTIFICATION OF FEATURES OF YOUR PRODUCT

Having researched similar products, and gained some target market feedback, decisions need to be made about which features will be included and which will be discarded, including new, original ideas. This is a synthesis of all thinking so far.

You should have a user that is representing your stakeholders. Seek their opinions. Conclude decisively which direction you will take with core mechanics of the game. e.g. health bar or lives?



For the highest marks in this section you can't simply list the features and limitations of your proposed solution, you must provide an explanation along with each one.

IDENTIFICATION OF LIMITATIONS AND SCOPE

No development studio can create everything they would like. They are limited by deadlines to release, money and programmer capacity. Be realistic about the time you have available. Whilst it might be great to include 100 levels with 5 types of player ship, 30 alien types and 20 power-ups, you may not have time for this!

This section is about 'scoping' your solution. Decide the parameters.

You may also be limited by your hardware and software. Perhaps you can only achieve the best product by using some of the paid-for libraries, and this will restrict your ambitions. For example, a Pyro framework includes dynamic lighting, shadow casting, tiling engine, particle effects and a better GUI. Without it, this is likely to be very hard work.

Think about some of the key restrictions that will have to be put on the game mechanics.

You may also be limited by your hardware. Maybe your graphics card cannot handle the frame rate or resolution you would like. You may have to simulate the input and output of your target device during development, especially if you are targeting the tablet, mobile or console market.

COMPUTATIONAL METHODS

By the very nature of what they are, computer games are problems solvable by computational methods. You can't have a computer game without a computer! However, there are a number of thoughts that must be employed in order to create what is either a fantasy world, or an abstraction of reality.

ABSTRACTION AND VISUALISATION

- What are the key objects in your game?
- What sprites, symbols, icons or font sets will be needed?

THINKING AHEAD

- What will your inputs be?
- What will your output screens be?

THINKING PROCEDURALLY

- What 'game states' will be necessary?
- What happens in each of the game states?
- Pipelining: what order do things need to happen?

THINKING LOGICALLY

- What are the critical conditions in the game (these eventually become if statements in code)?
- Mention the 'main game loop': the key procedures that will loop continuously until the game is over.

THINKING CONCURRENTLY

Are there aspects of the game where more than one thing happens at once? The classic example is the sound player using multi-channel sound to play different sound effects and background music at the same time as updating the game logic.



For the highest marks in this section you **MUST** make sure to clearly **explain** why the problem you are solving is amenable to a computational approach. In other words, you have decided to write a computer game for your project. Not every problem can be solved computationally, there are problems out there for which a computer program based solution are not appropriate or indeed possible, yours is, explain this.

HARDWARE AND SOFTWARE REQUIREMENTS

IDENTIFICATION OF THE HARDWARE REQUIREMENTS

There are hardware requirements for the development, and hardware requirements for deployment.

DEVELOPMENT LANGUAGE

Which type of computer is being used for the development?

How much memory does it need?

How much disk space does it need?

DEPLOYMENT

These are the minimum hardware requirements to run on the target platform.

How much disk space does your executable game require?

What specific input and output devices will be needed?

You should state the development environment requirements and the minimum specification required for the player.

IDENTIFICATION OF THE SOFTWARE REQUIREMENTS

The development environment will require an operating system.

Additionally there are different software requirements depending on your target. Here are some that may be required, but it depends on your development language or software development kit (SDK).

WINDOWS DESKTOP

OpenGL API for graphics rendering

OpenAL API for audio

MinGW 4.8.1 or Microsoft Visual C++ Express 2010 (MSVC)

OpenAL Windows drivers

ANDROID

Android SDK. You only need the "SDK Tools" version, not the "Eclipse+ADT Plugin" one

Java SE JDK 32bit version

Apache Ant Java library

IOS

Apple Mac computer running OS X

XCode developer tools with iOS SDK

HTML5 BROWSER TARGET

An HTML5 capable browser such as Chrome, Firefox, Opera, Safari, Edge or IE

IDENTIFICATION OF UTILITIES/LIBRARIES

Many languages have a core syntax model, but require additional libraries to be included for specific functionality such as file handling and networking.

Consider which libraries need to be included for your game to run.

Many games development languages also have third party modules that can be plugged in such as:

Box 2D - a common physics engine.

Spline - used for advanced animations.

Pyro - used for dynamic lighting, cameras, tile systems and particle engines.



For the highest marks in this section you need to make sure you have **justified** the hardware and software requirements you have listed. It is not sufficient to say your program will require 4Gb of RAM, a minimum of Windows 10 and 250Mb of hard drive space for example, you must explain why you have come up with these figures.

SUCCESS CRITERIA

IDENTIFICATION OF THE SUCCESS CRITERIA (REQUIREMENTS SPECIFICATION)

The requirements specification serves three purposes.

1. It is an agreement about the scope of the solution: exactly what will be included.
2. It provides a framework for testing the solution against.
3. It provides a reference for evaluation.

It is therefore very important that the success criteria are specific and measurable.

A good approach is to use numbered points.

Statements such as, "must be easy to use" are too subjective. This should detail exactly how this would be achieved, rather than being a broad statement.

Analysts will consider 'volumetrics': how many, how often. Using numbers makes your objectives specific and easily measurable.



For the highest marks in this section you must make sure you **justify** each of your success criteria / requirements. You can't simply make a list of them. Why does your program have this requirement? Where did it come from? Was it a user interview or another part of your research which made you come up with this requirement?

EVIDENCE DO'S AND DON'TS CHECKLIST

This section of your project write up **must** include:

SECTION	DO's	DON'Ts
A description of the game	1 Provide an outline of what your game is about 2 Provide an explanation of features required in your computer game	<input type="checkbox"/> <input type="checkbox"/> Reply on simple statements of the problem
Identify all stakeholders	3 Identify all the stakeholders (users) as individuals, groups or persona 4 Keep returning to the stakeholders (users) for input throughout the project	<input type="checkbox"/> <input type="checkbox"/> Identify an end user who cannot be easily contacted throughout the project
Justify why the game can be solved by computational methods	5 Explain why writing a computer game is suited to a computer program 6 Explain the features of your game that are amenable to a programmed solution 7 Explain why the output from the game is valuable to the stakeholders (users), a game should be interesting, make the use want to carry on playing!	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> Simply state that you are going to create a game because it is needed. You must justify decisions
Research	8 Provide detailed research into existing games which are similar (e.g. same genre) 9 Show that the research identifies features that can be adapted for use in your proposed game 10 Show how the research provides insight into the proposed game and how the features to be used are appropriate	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> Rely on your own input for the solution to your game Rely on an interview with an end user for all your research into the game
Features of the proposed solution	11 Identify the features of your proposed game 12 Identify any limitations of your proposed game 13 Be realistic about what can be achieved in the time allowed!	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> Attempt to write a game that is too complex in the time allowed
Software and hardware requirements	14 Specify any hardware requirements for your game 15 Specify any software requirements for your game 16 Do identify any additional utilities that will be required to implement your game	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> List all the software available simply to justify a choice Simply identify what software you are using
Success criteria	17 Specify the success criteria (requirements) for your proposed game 18 Do specify success criteria (requirements) that can be demonstrated through testing	<input type="checkbox"/> <input type="checkbox"/> Specify vague subjective criteria, such as colorful interface or easy or quick to use

DESIGN (MAX 15 MARKS)

1-2 marks

- Described elements of the solution using algorithms.
- Described some usability features to be included in the solution.
- Identified the key variables / data structures / classes (as appropriate to the proposed solution).
- Identified some test data to be used during the iterative or post development phase of the process.

5-8 marks

- Broken the problem down systematically into a series of smaller problems suitable for computational solutions describing the process.
- Defined the structure of the solution to be developed.
- Described the solution fully using appropriate and accurate algorithms.
- Described the usability features to be included in the solution.
- Identified the key variables / data structures / classes (as appropriate to the proposed solution) and any necessary validation.
- Identified the test data to be used during the iterative development of the solution.
- Identified any further data to be used in the post development phase.

9-12 marks

- Broken the problem down systematically into a series of smaller problems suitable for computational solutions explaining the process.
- Defined in detail the structure of the solution to be developed.
- Described the solution fully using appropriate and accurate algorithms explaining how these algorithms form a complete solution to the problem.
- Described, explaining choices made, the usability features to be included in the solution.
- Identified and justified the key variables / data structures / classes (as appropriate to the proposed solution) explaining any necessary validation.
- Identified and justified the test data to be used during the iterative development of the solution.
- Identified and justified any further data to be used in the post development phase.

13-15 marks

- Broken the problem down systematically into a series of smaller problems suitable for computational solutions, explaining and justifying the process.
- Defined in detail the structure of the solution to be developed.
- Described the solution fully using appropriate and accurate algorithms justifying how these algorithms form a complete solution to the problem.
- Described, justifying choices made, the usability features to be included in the solution.
- Identified and justified the key variables / data structures / classes (as appropriate to the proposed solution) justifying and explaining any necessary validation.
- Identified and justified the test data to be used during the iterative development of the solution.
- Identified and justified any further data to be used in the post development phase.

In this section you are discussing **how** you are going to solve the problem.

It should be the approach to be taken to implement the points from the requirements specification. An experienced programmer should be able to pick up your design, understand what is required, and code the solution in any suitable language without the need for further discussion.

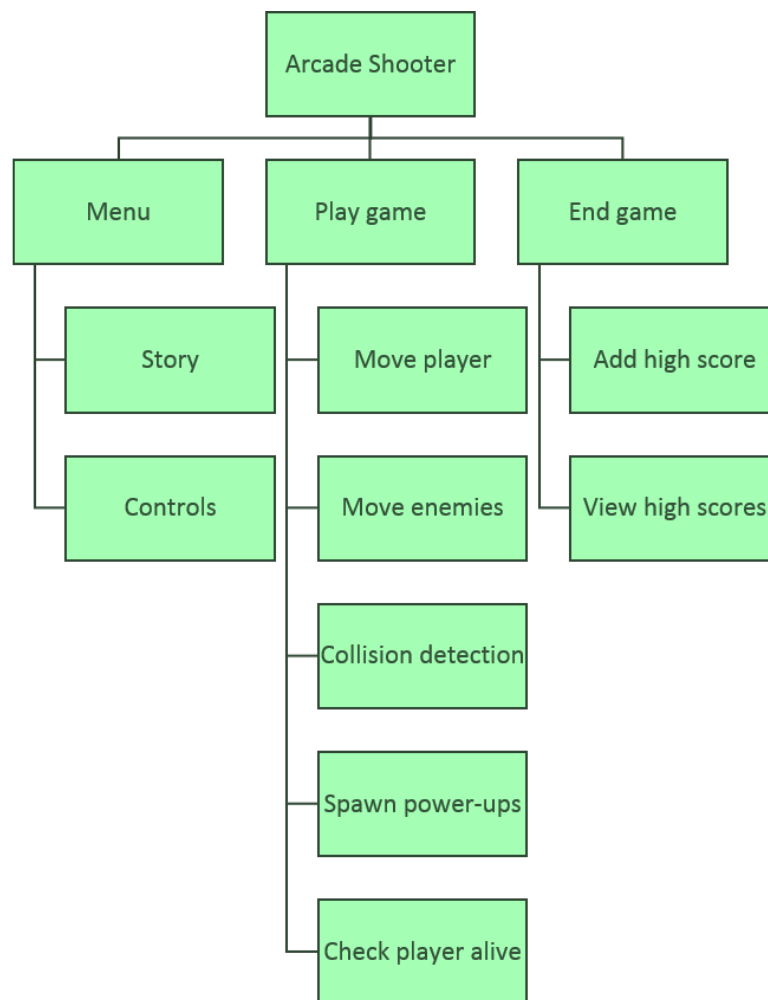
Strongest projects would include:

STRUCTURE OF THE SOLUTION

AN OVERVIEW OF THE STRUCTURE OF THE SOLUTION (A SYSTEMS DIAGRAM)

It is helpful to show an overview of the key modules of the project as an introduction and reference point for the rest of the design section.

Here is part of an example for a typical arcade shooter game. Note, this is not complete, but gives you an indication of the level of detail and structure required here. It is fine to use 'Shapes' in Office applications to create this diagram.



The diagram should fulfil all the points set out in the requirements specification, but without too much detail. Just the name of a method, procedure or function would be all that is required. This diagram does not show the classes or data structures, just the actions that need to be programmed.

This diagram will also serve as a reference for the development story, and it should be easy to cross reference the rest of the project to aspects of this diagram.



For the highest marks in this section you should **explain AND justify** the process you have taken to breaking your computer game down. Why are you using this approach to help you design the game solution? Think about key terminology you have learnt in the course to use here such as “Thinking Logically”, “Thinking Ahead”, Top-down Module Design”, “Step-wise Refinement”, “Decomposition” etc.

DECOMPOSITION OF THE PROBLEM

EXPLANATION OF EACH OF THE MODULES, METHODS, PROCEDURES AND FUNCTIONS REQUIRED

Take each of the boxes identified in the systems diagram, and in turn:

- Name the procedure, method or function.
- Explain the purpose of the section of code to be written: what it aims to achieve.
- Illustrate any graphics, or describe sounds that will be used.

An example of an extract for this section: Procedure `read_level`

All of the levels are stored in text files. The file name represents the level being played, e.g. '1.lv' is level 1, '2.lv' is level 2. In the text file, a character represents each separate object in the game:

A - enemy type A

B - enemy type B

C - asteroid

1 - health boost power-up

To show the level on the screen it first has to be read into an array, which will be the same length as the text file. The entire level will be read once, and held in memory whilst being played to ensure no disk reads are required whilst playing which could affect the frame rate. Using a text file on disk, and an array in memory to represent a level, makes it easy to design levels with a simple text editor.

Each character is read into the array by reading in one entire line of characters and then splitting it down one by one before moving onto the next line. The start position of the object in the level is determined by its row, representing time, and its column, representing its horizontal position on-screen.



For top marks make sure to **justify** each of the usability features you are talking about. Why are you including them? How does their inclusion make your game intuitive to use and play?

A JUSTIFICATION OF THE APPROACH TAKEN

Note how the approach was justified when explaining the decomposition of the problem:

"The entire level will be read once, and held in memory whilst being played to ensure no disk reads are required whilst playing which could affect the frame rate. Using a text file on disk, and an array in memory to represent a level, makes it easy to design levels with a simple text editor."

Where appropriate you should explain the design decisions you are making. There is often a compromise between elegance in code and readability for future developers. Or between file storage and processing requirements.

With games development, some of the key goals will be:

- A high frame rate achieved through efficient processing and suitable algorithms.
- Graphics fidelity: the art concept of the game, e.g. low polygon or high polygon. Untextured or textured environments. 2D or 3D, first person or third person. Considerations about the skybox, and background art.
- Sound, creating emersion.
- Level design creating a progression of challenge and slow introduction of game objects/mechanics for the player.
- Readability of code/files: being able to easily add and change how the game plays. Adding new content, and being able to easily balance game mechanics later.

Think about how your procedures, methods and functions achieve these goals.

KEY VARIABLES AND STRUCTURES

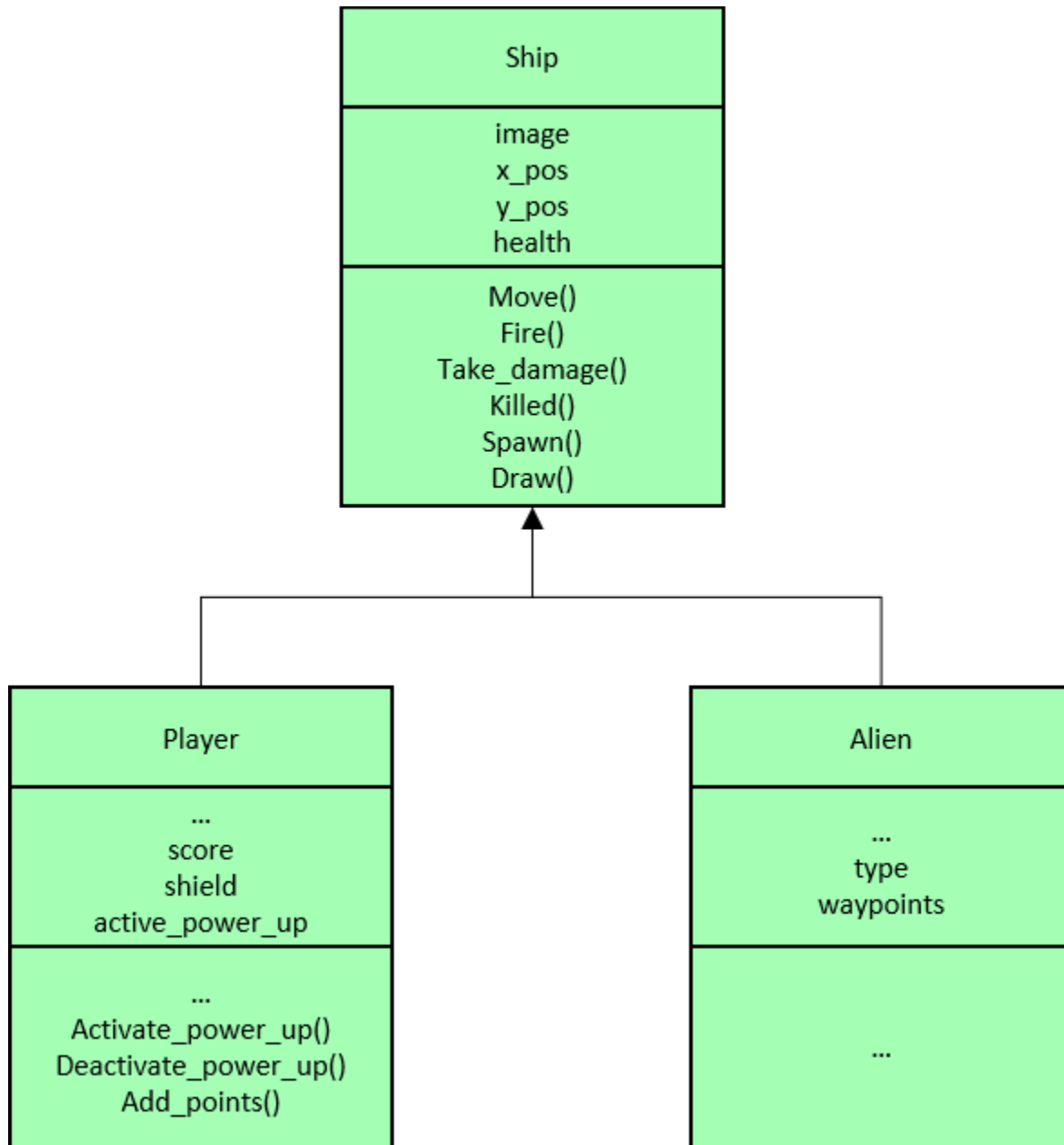
IDENTIFICATION OF THE KEY CLASSES AND ATTRIBUTES IF OBJECT ORIENTED METHODS ARE USED

Ideally your project will use object oriented techniques. Therefore you should include class diagrams to state the names of the classes, attributes and methods.

The names of the methods should match what you have written in the previous section for decomposition of the problem.

If further explanation of techniques is required here, you should include it. For example, the reason for using inheritance may not be obvious from what you have written so far.

EXAMPLE CLASS DIAGRAM



EXPLANATION AND JUSTIFICATION OF THE DATA STRUCTURES: ARRAYS/LISTS AND FILES

This section may well already be covered in the decomposition of the game, depending on your approach to the write-up.

You should include lists of:

- Global variables: name and purpose.
- Data structures used, e.g. arrays, linked lists, trees etc, and their purpose including encoding.
- Data files and their structure/encoding.

Encoding is the meaning behind the raw data. For example, in a break-out game, a 1 may represent a yellow tile, a 2 a green tile, 3 a non-breakable tile, 4 a two hit tile etc.

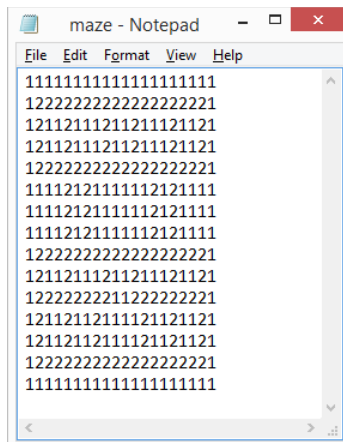
If your game includes lots of similar files, e.g. five files for five level designs, then it would be appropriate to illustrate just a couple of these.

If you are using sprite sheets then it is appropriate to show them in this section, and explain how they are accessed and divided at run-time.

EXAMPLE OF A DATA FILE AND THE ENCODING

The data for each maze is encoded in a text file called maze[level number].txt.

Example of a typical maze file:



```
11111111111111111111
12222222222222222221
12112111211211121121
12112111211211121121
12222222222222222221
11112121111121211111
11112121111121211111
11112121111121211111
12222222222222222221
12112111211211121121
12222222211222222221
12112112111121121121
12112112111121121121
12222222222222222221
11111111111111111111
```

A '1' represents a wall that the player cannot walk through, and a '2' a sandy passage that the player can walk along. These are stored in this format for the screen render method to draw the appropriate tile.

This data is read into a 2D array called maze when a new level is started.

The game uses a sprite sheet to store the graphics in one file:



Each tile is 32x32 pixels. Larger objects such as houses, are made up of multiple tiles.

The screen resolution is 640x480 pixels. The maze is 20 tiles wide, and 15 tiles tall (20*32=640, 15*32=480).



For the highest marks it is **VERY** important in this section that you fully identify any key variables and data structures you intend to use. Make sure to name the **procedures, functions, methods, classes/objects, arrays, structures, records, files** etc. and any key **variables/constants** along with their **datatype**.

For example, a high score table may be stored in memory as an array and on a disk in a sequential file. For games with levels, include one example of a text file that will be read in for the level data.

ALGORITHMS

A SET OF ALGORITHMS IN PSEUDOCODE TO DESCRIBE EACH OF THE METHODS/FUNCTIONS

This is arguably the hardest section to complete, as you probably have very little experience to date of writing the type of game you are undertaking. The best way of approaching it is to take each of the boxes on your systems diagram and think carefully about the step by step actions that have to take place for that part of the program to work.

This is not the actual code, but a language independent approach to solving the problem. Done correctly, a programmer would be able to write actual code from your algorithms in any language they are familiar with.

You can use flowcharts, but these are likely to be very cumbersome and time consuming to produce. A better approach is to use pseudocode.

Every method, function, and procedure should be presented.

You may well not have a full idea of how parts of your game will actually be implemented at this stage. Now is the time to give it some thought. It is fine to change your mind later in the development story.

Example pseudocode for moving a player object:

Procedure Move_player:

Routine that moves the player on the screen when an input is made.

1. If the key pressed is left arrow then:
2. Check if the player x position is greater than 0:

3. If it is, reduce the x position by 5 pixels
4. If the key pressed is right arrow then:
5. Check if the player x position is less than screen width:
6. If it is, increase x position by 5 pixels

You will see from this example, it is useful to also indent the pseudocode.

The algorithms should match the requirements specification in the analysis section, and the decomposition of the problem in the design section. An examiner should be able to "follow the thread" of small aspects of your project from the initial research, through to requirements, through to design, to algorithms, how these were developed, tested, and evaluated. Make sure everything is coherent and easy to follow throughout your project write-up.



For the highest marks in this section it is important at some point (probably towards the end of your design) to **justify** how the set of algorithms you have presented form a complete solution to the your problem.

EXPLANATION AND JUSTIFICATION OF THE DESIGN OF THE USER INTERFACE

Computer games will typically have 3 main screens: a menu, game-playing and end-game, although there could be more. This section should show the proposed screen designs, and explain how they make the game easy to play and understand for the user.

For example, first person shooter games usually have a HUD, markers, ammo indicators, kill streak indicators etc. These are all part of the user interface and their inclusion or otherwise should be justified. It would be a good idea to get some user feedback on these designs, and include that in the write-up of this section.

Example of a user interface and identification of items for further discussion:



From looking at this proposed game-playing screen, you can establish that the player has both a shield and a health bar. These could be discussed in terms of how they work. For example, if the player's health gets low, the red bar starts to flash. That is not obvious without further commentary. What are the active power-up indicators for? What are the different level object indicators? A programmer couldn't code your solution without understanding the requirements of the user interface.

DESCRIPTION AND JUSTIFICATION OF THE USABILITY FEATURES

In computer games, thought is given to the input mechanisms. Keyboard & mouse, controller support etc. This is an opportunity to discuss the input mechanics and the keys that the player would need to know. With very simple games there is a choice between using WSAD and arrow keys. Space or Enter.

Thought is also given to those who are colour blind. What will you do to your interface to ensure the player is not disadvantaged? Having green and red markers on a HUD can often be changed to orange and blue in the options for example.

In some games there is an opportunity to select alternative control mechanisms, such as 'bumper-jumper'.

Here is a post from a community site explaining the purpose:

Bumper Jumper made its "debut" in Halo 3. It became a quickly adapted favourite of competitive players because you could now jump AND aim (since your thumb doesn't have to leave your right stick) which can be very beneficial in winning your individual gunfights. It developed a tactic known as a jump strafe, which was often paired with "Gandhi hopping" mid-air to avoid dying. Players would do anything to not die in those crucial moments, thus Bumper Jumper quickly became a new favourite layout amongst players. I'd also like to note that because of bumper jumper, you were also able to more easily aim your melee's while mid-air... which could help you trade kills, or in more flashy situations you could more effectively "ninja" players with a well-timed jump followed by an aimed back smack.

Previously the only way to achieve "jump shots" pre-Halo 3 was to play using a grip known as "claw". This is when you fire the right trigger by using your right middle finger, and placing your right index finger over the top of your ABXY face buttons. Claw is very advantageous, because a player could essentially switch weapons, reload, jump, and often melee while still never having to leave the right stick, occurring little to no loss in aim. Playing claw carries over many benefits amongst several shooter titles. It's used in Halo, CoD, and even GoW... however, it has been said that over time it can damage your hand. Based on that notion alone, it has allowed companies like SCUF gaming to thrive by creating controllers to perform actions on the back of controllers via a "paddle" system. Typically, there are 2 paddles on the back of SCUF controllers, but they can be configured for up to 4 paddles. What are paddles? They are elongated buttons that can be used to replace the functions of any button on your controller, not just ABXY. These controllers are used by literally every single professional Call of Duty player.



For top marks make sure to **justify** each of the usability features you are talking about. Why are you including them? How does their inclusion make your game more intuitive and easier to use?

DESCRIPTION AND JUSTIFICATION OF THE VALIDATION REQUIRED

Validation (checking the user has given a valid input) in computer games is very different to other projects. Here the player has very limited control over input, and therefore much validation is not required. However, there are a few instances to consider. For example:

- How is the player prevented from leaving the screen/map?
- Is there a restriction to the number and type of characters on a high score table?
- Are there only certain buttons that function on the input screens? For example, X to skip during the story and cut-scenes.

IDENTIFICATION AND JUSTIFICATION OF TEST DATA TO BE USED DURING DEVELOPMENT

Sometimes referred to as bottom-up testing, as each section from the systems diagram is developed, it needs to be tested for functionality before moving on to the next stage.

For each of the modules, methods, procedures and functions described in the decomposition of the problem section, you should explain the testing that would take place, and the data that would be used.

If you prefer, you can include this at the end of each description of the modules, methods, procedures and functions rather than a separate section in your write-up.

The easiest way to structure this work is in a series of small tables. Do not do this in one big table, it is too difficult to follow.

Example of bottom-up testing for the player ship moving left and right:

What is being tested	Data input	Expected output
The player can move left	Holding down the left arrow key or A.	The player ship moves to the left and stops at the edge of the screen.
The player can move right	Holding down the right arrow key or D.	The player ship moves to the right and stops at the edge of the screen.
The speed of movement: normal	N/A	The player should be able to move from the far left to the far right in 3 seconds with no power-ups.
The speed of movement: power-up	Collected speed-up power-up.	The player should be able to move from the far left to the far right in 1.5 seconds with no power-ups.
The speed of movement: power-up	Collected speed-up power-up.	The power-up should stop after 20 seconds.

IDENTIFICATION AND JUSTIFICATION OF THE TEST DATA TO BE USED POST-DEVELOPMENT TO ENSURE THE SYSTEM MEETS THE SUCCESS CRITERIA

Sometimes referred to as alpha testing, this is about making sure the complete development meets the requirements specification. For computer games, much of this will be complete in the development testing, but there are usually other factors to consider once the development is complete. The game will need balancing, it will need to be fun, and progression will need to be appropriate. With big multiplayer games, developers are keen to test what happens in big multi-user situations to stress test the servers and check they spin up and close down as the number of online players grows and shrinks during the day.

You will not be doing anything as grand as this, but some of the points will be appropriate. It is often not easy to test how the game feels until the development is complete.

An approach here is to take each of the points from the requirements specification and draw up a similar table that explains how they will be tested for functionality. This is an opportunity to include those tests that are less measurable, and are more about the feel of the game. You will have omitted these in the requirements specification because they are subjective. For example, the game should be fun. How will you test and measure this?

IDENTIFICATION OF DATA THAT IS DESIGNED TO TEST THE ROBUSTNESS OF THE SOLUTION

Good testing should attempt to break the program. There is nothing additional to write in this section. In the testing tables you have already created, you should have ensured that there were tests designed to break the game.

For example, have you stress tested the frames per second when lots of objects are on-screen in the rendering routine? Have you checked what happens with invalid keyboard inputs?



For the highest marks in this section make sure to **justify** the test data you plan to use during development, it is not sufficient simply to state you will be using certain values.

EVIDENCE DO'S AND DON'TS CHECKLIST

This section of your project write up **must** include:

SECTION	DO's	DON'Ts
Decompose the problem	19 Provide evidence of decomposing your game into smaller problems	<input type="checkbox"/> Simply state the game as a single process
	20 Provide evidence of a systematic approach, explaining and justify each step in the process	<input type="checkbox"/>
Structure of the solution	21 Provide a detailed overview of the structure of your game	<input type="checkbox"/>
Algorithms	22 Provide a set of algorithms to describe each of the sub-parts of your game	<input type="checkbox"/> Simply provide an outline data flow
	23 Show how the algorithms fit together to form a complete working game	<input type="checkbox"/>
	24 Show how the algorithms have been tested to show that they worked as required	<input type="checkbox"/> Provide code or reverse engineered code as an algorithm
Usability features	25 Describe with justification the usability features of your proposed game	<input type="checkbox"/> Spend ages creating colorful diagrams of the game
	26 Explain and justify the design of any user interface / screen designs or interfaces with other systems	<input type="checkbox"/>
Key variables and structures	27 Identify and justify the key variables	<input type="checkbox"/>
	28 Explain and justify the data structures that are to be used in your game	<input type="checkbox"/>
	29 Describe and justify any validation required	<input type="checkbox"/>
Test data for development	30 Identify and justify any test data to be used during development (this is appropriate test data that can be shown to test the functionality of your game for development testing purposes)	<input type="checkbox"/> Create a full test plan for this stage; this is data to be used at each stage of the development process
Test data for beta testing	31 Identify and justify test data to be used post-development to ensure the game meets the success criteria (requirements)	<input type="checkbox"/> Create a test plan for this at this stage; the data will be used in a final test plan for the product at the post-development testing stage
	32 Identify data that is designed to test the robustness of the game; good testing should attempt to break the game	<input type="checkbox"/>

DEVELOPING A CODED SOLUTION (MAX 25 MARKS)

ITERATIVE DEVELOPMENT OF A CODED SOLUTION (MAX 15 MARKS)

1-4 marks

- Provided evidence of some iterative development for a coded solution.
- Solution may be linear.
- Code may be inefficient.
- Code may not be annotated appropriately.
- Variable names may be inappropriate.
- There will be little or no evidence of validation.
- There will be little evidence of review during the development.

5-8 marks

- Provided evidence for most stages of the iterative development process for a coded solution describing what they did at each stage.
- Solution will have some structure.
- Code will be briefly annotated to explain key components.
- Some variable and/or structure names will be largely appropriate.
- There will be evidence of some basic validation.
- There will be evidence that the development was reviewed at some stage during the process.

9-12 marks

- Provided evidence of each stage of the iterative development process for a coded solution relating this to the break down of the problem from the analysis stage and explaining what they did at each stage.
- Provided evidence of some prototype versions of their solution.
- The solution will be modular in nature.
- Code will be annotated to explain all key components.
- Most variables and structures will be appropriately named.
- There will be evidence of validation for most key elements of the solution.
- The development will show review at most key stages in the process.

13-15 marks

- Provided evidence of each stage of the iterative development process for a coded solution relating this to the break down of the problem from the analysis stage and explaining what they did and justifying why.
- Provided evidence of prototype versions of their solution for each stage of the process.
- The solution will be well structured and modular in nature.
- Code will be annotated to aid future maintenance of the system.
- All variables and structures will be appropriately named.
- There will be evidence of validation for all key elements of the solution.
- The development will show review at all key stages in the process.

TESTING TO INFORM DEVELOPMENT (MAX 10 MARKS)

1-2 marks

- Provided some evidence of testing during the iterative development process.

3-5 marks

- Provided some evidence of testing during the iterative development process.
- Provided evidence of some failed tests and the remedial actions taken.

6-8 marks

- Provided evidence of testing at most stages of the iterative development process.
- Provided evidence of some failed tests and the remedial actions taken with some explanation of the actions taken.

9-10 marks

- Provided evidence of testing at each stage of the iterative development process.
- Provided evidence of any failed tests and the remedial actions taken with full justification for any actions taken.

This should **NOT** be a separate section in your report, this should form an integral part of “The Development Story”. You can therefore see your development story as providing you with evidence for all of the 10 marks above. Read the mark scheme carefully above for “Testing to inform development” and make sure this is covered during “The Development Story”.



To get full marks in this section we should be seeing testing going on throughout the development story and if any of your tests fail then it is **ESSENTIAL** their follows evidence of what you had to do next. This should happen quite often, no one codes perfectly the first time, tests should and will fail. Explain what happened, show what you did to address this and **justify** your actions.

In this section you are discussing **how you** have actually **solved the problem**.

It should be a story of how the key components of the product were produced, how you had to change your plans along the way, and the testing of those components. You can't document everything, so these will be the key milestones and decisions.

Using the systems diagram for reference, identify about 6-8 key milestones in your development. For example, in Pacman, this could be:

- Drawing the maze and the pills.
- Moving pac-man around the maze.
- Collecting the pills and starting a new level once collected.
- Moving ghosts around the maze.
- Adding collision detection for pac man and the ghosts.
- Adding power-pills and changing ghost behaviours.
- Polishing the game: scoring, special effects, sound and menus.

Strongest projects would include:

ITERATIVE DEVELOPMENT

Evidence of how the program was developed stage by stage. This should be a "development story", not just a code dump. Typically it might include:

- Explanation of how each section of the program was coded.
- Modular, commented code, with variables appropriately named.
- Prototype versions of the program at each stage of the process.
- Evidence of validation.
- Bottom-up testing covering a wide range of valid, invalid inputs and situations.
- Review of each stage of the process.
- Explanations of any changes required and any modifications to the design of the solution that result from the testing, including remedial actions.

It is perfectly acceptable for the design to change as the program develops. There is no need to go back and change the design section, just document the changes and amendments to the success criteria as part of the iterative development process.

Example of a small part of this section:

Class Rocket

```
Field sprite:Image = LoadImage ("player.png")
```

```
Field x:Float = 300
```

```
Field y:Float = 420
```

```
Method Move(x_distance:Int)
```

```
    x+=x_distance
```

```
    If x<0 Then x=0
```

```
    If x>590 Then x=590
```

```
End
```

```
End
```

```
Field player:Rocket
```

```
Select GameState
```

```
Case "PLAYING"
```

```
    'Handle inputs
```

```
    If KeyHit (KEY_ESCAPE) Then GameState="MENU"
```

```
    If KeyDown(KEY_LEFT) Then player.Move(-10)
```

```
    If KeyDown(KEY_RIGHT) Then player.Move(10)
```

```
    If KeyHit(KEY_P) Then SetMusicVolume(0.1)
```

```
    If KeyHit(KEY_O) Then SetMusicVolume(1)
```

```
    If KeyHit(KEY_M) Then
```

```
        If MusicState = 1 Then
```

```
            PauseMusic
```

```
        Else
```

```
            ResumeMusic
```

```
        End
```

```
End
```

This part of the code allows the player to move their ship left and right using the cursor keys. Additionally a player can press escape to return to the main menu, quitting the game at any time.

In the playing game state I added a section to handle all the inputs. If escape is pressed the game state returns to a menu state, effectively ending the game. However, the variables will not be reset, so this could in fact be used as a pause option instead. The variables are reset when the game is started again from the menu.

I created a new class called rocket, and an instance of the class called player. The image of the space ship is loaded when the instance is created. The x and y position attributes puts the ship in the middle at the bottom of the screen.

When the left or right key is pressed it passes the number of pixels to move the ship into the rocket move method. This number is positive to move the ship right, or negative to move the ship left.

The movement is clipped at the left and right edge of the screen, creating suitable validation of the input.

I decided to add an option to toggle the background music on and off using the M key, as well as making the music quieter relative to the sound effects by pressing P to reduce the volume, and O to increase the volume.

Originally I wasn't going to have background music, but I found this added to the experience of the game. However, the user said that it was sometimes annoying and wanted to be able to toggle it on and off. I thought it would also be good to turn the volume up and down. This could be achieved with a slider, or number input in a menu screen, but this would be a lot more code, so a good compromise was to be able to make the music 1/10th of the volume, making it a lot quieter than the sound effects to be programmed later.

Test condition	Inputs required	Expected output	Review and remedial action required
Player can move left.	Left cursor key	Player moves left at a controllable speed and stops at the left edge of the screen.	Player moved a little too slowly. Increase number of pixels moved.
Player can move right.	Right cursor key	Player moves right at a controllable speed and stops at the right edge of the screen.	Player moved a little too slowly. Increase number of pixels moved.

Game can be quit at any time	Escape key	Game returns to the main menu and background music stops.	Game restarted from previous position when replayed. Variables need to be reset for a new game.
Music toggles on and off	Escape key	Music turns off if it is playing and on if it is not playing.	None.
Music volume can be increased and decreased	O key P key	Music volume increases. Music volume decreases.	None.



For the highest marks in this section the key is make sure everything you do is **explained AND justified** and that you don't miss anything out! Each stage of your iterative development should be linked back to your break down that you did in the analysis & design phase, it must not seem as if your development and your previous planning work is disconnected. Make sure that any code you write is well annotated with coder comments, candidates often forget this and make sure that variable, constant, procedure, function and methods all have sensible and meaningful names.

EVIDENCE DO'S AND DON'TS CHECKLIST

This section of your project write up **must** include:

SECTION	DO's	DON'Ts
Iterative development	33 Provide evidence of iterative development showing how the complete game was developed stage by stage "The development story"	<input type="checkbox"/> Simply supply completed code for the program as evidence
	34 Provide evidence showing how each section of the game was coded and tested	<input type="checkbox"/>
Prototyping	35 Provide prototype versions of the game at each stage of the process that show the annotated and explained code	<input type="checkbox"/>
	36 Provide evidence of testing at each stage using the test data identified in the design section	<input type="checkbox"/>
Annotated modular code	37 Annotate the code at each stage of the process	<input type="checkbox"/> Simply supply the complete code for the program as evidence; the code must be developed in suitable stages
	38 Use meaningful names for all variables, structures and modules	<input type="checkbox"/>
	39 Provide code in a modular form	<input type="checkbox"/>
	40 Provide the code as separate modules	<input type="checkbox"/>
Validation	41 Supply evidence of validation	<input type="checkbox"/>
	42 Supply evidence that the validation has been tested and works as expected	<input type="checkbox"/>
	43 Supply evidence that all testing covers a wide range of valid and invalid inputs and situations	<input type="checkbox"/>
Reviews	44 Review each stage of the process in the development phase, summarizing what has been done and how it was tested	<input type="checkbox"/>
	45 Explain any changes required and any modifications to the design of the solution that result from the testing	<input type="checkbox"/>

EVALUATION (MAX 20 MARKS)

TESTING TO INFORM EVALUATION (MAX 5 MARKS)

1 mark

- Provided evidence of some post development testing.

2 marks

- Provided evidence of final product testing for function.

3-4 marks

- Provided annotated evidence of post development testing for function.
- Provided annotated evidence for usability testing.

5 marks

- Provided annotated evidence of post development testing for function and robustness.
- Provided annotated evidence for usability testing.

TESTING

You need to prove to the examiner that your solution works. The best way of achieving this with a minimal amount of additional work is to:

- Copy into this section, the test table you created in the design section (Identification and justification of the test data to be used post-development to ensure the system meets the success criteria).
- Create a video screencast of the game being played through the test scenarios.

You can use trial versions of software such as Camtasia to achieve this.

TEST TABLE

In the test table, add an additional column to state whether the requirement was met or not, with an explanation of why criteria were partially, or not achieved. For criteria that were achieved, there should already be sufficient evidence in the development story about how these were achieved. If additional details are required, they can be added here.

VIDEO EVIDENCE

The video should walk the examiner through the tests with an audio commentary.

Part of a typical script may look like this:

"Test 1. In this test I am checking that the main menu is displayed, and that the player can switch to the controls and lore screens.

Test 2. In this test I am checking that the game screen is displayed and all the characters are spawned as expected.

Test 3. In this test I am checking that the player can move their ship left and right without going off the edge of the screen."

There is no need to write-up the script or submit it. You may find it a useful prompt when recording the video.

There is no upper limit to the number of tests that you should perform. They need to be comprehensive, but not trivial. There is no need to test every single button if they are all very similar for example. Navigation tests are less important than game logic tests. As a rough guide, you may be aiming for about 30 tests in total.

USABILITY FEATURES

ILLUSTRATION OF HOW THE USABILITY FEATURES HAVE BEEN TESTED TO MAKE SURE THEY MEET THE USER NEEDS

Either during the video, or as a section at the end, you should showcase how you have made the product usable. Some ideas to consider are:

- Clear prompts about which button to press to start the game, view the controls etc.
- Familiar, well recognised icons, matched to the genre.
- Player indicators such as score, number of lives etc.
- Familiar layout controls: cursor keys, WSAD, or keyboard mapping options for the player.



For the highest marks in this section it is vital there is clear evidence of **BOTH** post development testing **AND** usability testing. Break the two types of testing out and make it explicit which is which. The usability testing should be done by, or in the presence of, your end user.

EVALUATION OF SOLUTION (MAX 15 MARKS)

1-4 marks

- Commented on the success or failure of the solution with some reference to test data.
- The information is basic and communicated in an unstructured way. The information is supported by limited evidence and the relationship to the evidence may not be clear.

5-8 marks

- Cross referenced some of the test evidence with the success criteria and commented on the success or otherwise of the solution.
- Provided evidence of usability features.
- Identified some limitations on the solution.
- The information has some relevance and is presented with limited structure. The information is supported by limited evidence.

9-12 marks

- Used the test evidence to cross reference with the success criteria to evaluate the solution identifying whether the criteria have been met, partially met or unmet.
- Provided comments on how any partially or not met criteria could be addressed in further development.
- Provided evidence of the usability features.
- Considered maintenance issues and limitations of the solution.
- There is a line of reasoning presented with some structure. The information presented is in the most part relevant and supported by some evidence.

13-15 marks

- Used the test evidence to cross reference with the success criteria to evaluate the solution explain how the evidence shows that the criteria has been fully, partially or not met in each case.
- Provided comments on how any partially or unmet criteria could be addressed in further development.
- Provided evidence of the usability features justifying their success, partial success or failure as effective usability features.
- Provided comments on how any issues with partially or unmet usability features could be addressed in further development.
- Considered maintenance issues and limitations of the solution.
- Described how the program could be developed to deal with limitations of potential improvements / changes.
- There is a well developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated.

In this section you are discussing **how effective** your solution was.

It should be a final holistic (alpha) test of the complete solution against the requirements specification. It is an opportunity to showcase the final product, and evaluate how well it solved the problem set out in the analysis.

Strongest projects would include:

EVALUATION

COMMENTARY ON HOW WELL THE SOLUTION MATCHES THE REQUIREMENTS

A final, honest review of your product.

In the games industry, this will often be a review from critics, outlining the positive and negative aspects of the game. Games review channels on YouTube will give you a good indication of the requirements here.

Consider asking a friend to provide you with a review.

Consider the criticisms from your game review. How could you address these if you had more time? What approaches would be required? If you could release subsequent content (DLC), what would that include? More levels, more power-ups, more weapons?

A reflection on the key changes that were required between the design and development stages should be outlined here. You have probably already documented these in the development story or testing sections, so a cross reference with page numbers, and a general summary will suffice.

Address any unmet criteria or features that might be useful, and how these might be approached.



For the highest marks in this section your evaluation should be linked back to your test evidence and your success criteria / requirements. In other words, there should be a clear story and link between the three e.g. You came up with a requirement back in the analysis stage, you then tested it in the last section to see if had been a success or not and now you are evaluating how successful you were in meeting it, and if it wasn't met, why not? Again, as with previous sections much of the work here must be **explained AND justified** for you to get top marks, it is insufficient to simply make statements or describe what has happened.

EVIDENCE DO'S AND DON'TS CHECKLIST

This section of your project write up **must** include:

SECTION	DO's		DON'Ts
Testing	46 Provide evidence of testing on the completed game	<input type="checkbox"/>	
	47 Provide evidence that the game functions as designed	<input type="checkbox"/>	
	48 Provide evidence that the game is robust and will not fall over easily	<input type="checkbox"/>	
	49 Cross-reference the test evidence against the success criteria (requirements) from the analysis section to evaluate how well the game meets these criteria	<input type="checkbox"/>	
Usability features	50 Show how the usability features have been tested to make sure they meet the stakeholder's (user's) needs	<input type="checkbox"/>	
Evaluation	51 Comment on how well the game matches the requirements	<input type="checkbox"/>	Comment on the development process and anything your learned or how much you enjoyed it
	52 Comment on any changes that were made to the design during the development stage	<input type="checkbox"/>	
	53 Comment on any unmet criteria or features that might be useful and how these might be approached	<input type="checkbox"/>	
Maintenance	54 Discuss future maintenance / possible updates of the game and any limitations in the current version	<input type="checkbox"/>	
	55 Discuss how the game might be modified to meet any additional requirements or changing requirements	<input type="checkbox"/>	
	56 Comment on the maintenance features included in the game and report	<input type="checkbox"/>	

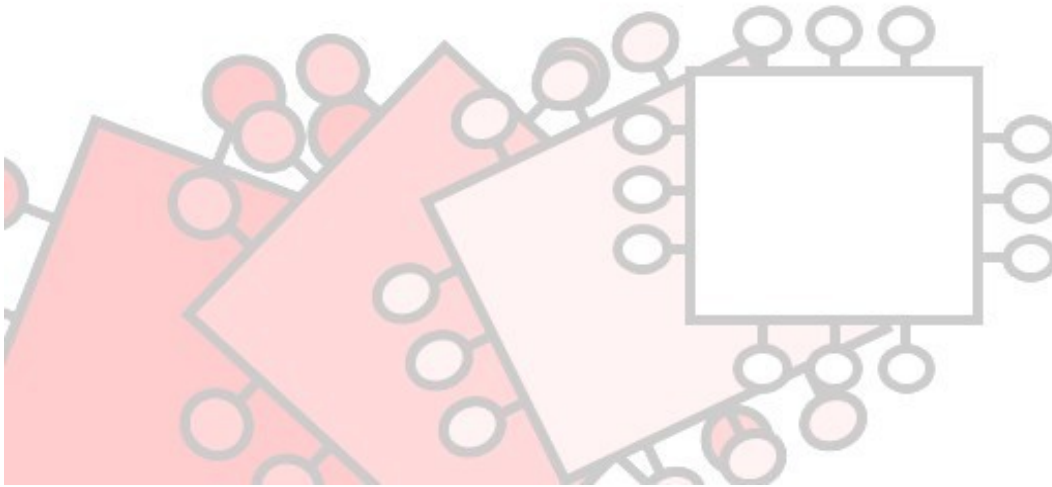
PROJECT HAND IN CHECKLIST

1. Make sure you project as a title page with:
 - a. "Title of Project"
 - b. The wording H446-03
 - c. Your full name
 - d. Your candidate number
 - e. Your centre name & number
2. Make sure your project contains a contents list.
3. Make sure every section of your project is included and clearly labelled:
 - a. Analysis
 - b. Design
 - c. Developing the coded solution ("The Development Story")
 - d. Evaluation
4. Make sure every single page is clearly numbered.
5. Make sure your project as a bibliography.
6. Make sure one of your Appendix's is clearly labelled as "Code Listings"
 - a. This should include a full print out of **ALL** code.
7. Make sure you have included a testing video.
8. Make sure you have included a copy of your entire solution on a CD, DVD or Pen Drive.

INITIAL PROJECT IDEAS

Use this area to record your initial ideas / thoughts for your Computer Science coursework project.

Your ideas will be used as the basis for writing the start of your “Analysis”.



A'Level OCR Computer Science H4406-03/04 Coursework Guide

For more help and resources, including appropriate project tutorials and guides for the Computing Project please visit craigndave.org